(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 653 716 A1**

(12) **EUROPEAN PATENT APPLICATION**

(21) Application number: **93402682.4**

(51) Int. Cl.[6]: **G06F 17/50**

(22) Date of filing: **02.11.93**

(43) Date of publication of application: **17.05.95 Bulletin 95/20**

(84) Designated Contracting States: **BE CH DE ES FR GB IT LI NL SE**

(71) Applicant: **BULL S.A.**
Tour BULL,
1, place Carpeaux
F-92800 Puteaux (FR)

(72) Inventor: **Tamisier, Thomas**
22, rue au Pain
F-78100 Saint Germain en Laye (FR)

(74) Representative: **Denis, Hervé et al**
Direction de la Propriété Intellectuelle BULL SA,
Poste courrier: LV59C18,
68 route de Versailles
F-78430 Louveciennes (FR)

(54) **Method of verification of a finite state sequential machine and resulting information support and verification tool.**

(57) The method of computing the reverse image of the transition function $\Delta(\delta, \delta')$ of a product finite state machine (PFSM) : $\Delta^{-1}(E_{n-1})$ from the set of n-1 equivalent states comprises the steps of (a) constructing in a canonical way, from the BDD of the graph of the equivalence relation E_{n-1}, the BDD of the graph of a total function from S into S, named cross-section and denoted $C(E_{n-1})$, (b) constructing from the cross-section and vector δ a new vector $\delta^{n-1} = C(E_{n-1}) \circ \delta$, and (c) computing the equivalent pairs of states with respect to the vector δ^{n-1} to have the pairs of $(\forall x \Delta^{-1}(E_{n-1}))$.

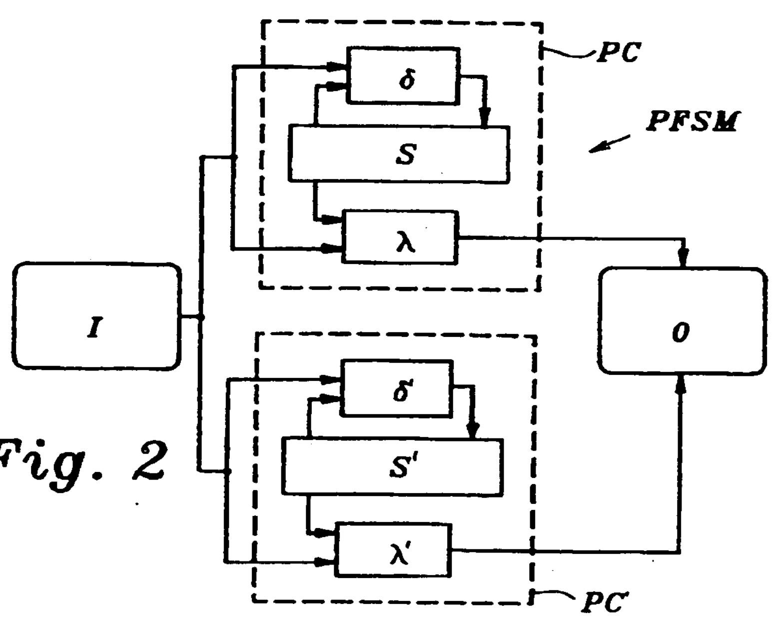

Fig. 2

EP 0 653 716 A1

The invention relates to a method of verification of a finite state sequential machine ad more particularly to a method of computing by means of a computer a equivalence relation of a finite state machine. It also relates to a information support incorporating a program carrying out the method of the invention and a verification tool carrying out the method.

The design of very large integration circuits requires zero-defect circuits because prototyping is very expensive to debug circuits. Hardware design and verification use an abstract description of a circuit realization (hardware device) and a circuit specification (expected behaviour) in the shape of a finite automaton, called finite state machine or FSM. The finite state machine is obtained from a description in an hardware description language (e.g., VHDL) of the circuit realization or the circuit specification by abstraction tools. The finite state machine performs the comparison by equivalence or implication of the circuit realization with the circuit specification. An example of a finite state machine is described in publication (1): Workshop on Automating Verification Method for Finite State Systems, Grenoble, July 1989, "Verification of Synchronous Sequential Machines Based on Symbolic Execution", Coudert et al. In a finite state machine is used a method of computing equivalence relations, in particular the observable equivalence. A present application of observable equivalence in hardware design and verification is mainly the tool called Auto/Autograph as described for example in Proceedings of the DIMACS Workshop on Computer-Aided Verification, Vol. 3, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 1990, pages 477-492, Roy et al. "Auto/Autograph". This tool allows a reduction of complexity of reachable states from the initial states while minimizing the number of state variables, which results in hardware design in a reduction of the number of state registers in a circuit.

In Figure 1 is a block diagram illustrating the operation of a finite state machine FSM. It uses three sets and two functions. The three sets comprise a set of inputs, named I, a set of states, named S, and a set of outputs, named O, and the functions comprise a transition function δ of the type $I \times S \rightarrow S$ and an output function λ of the type $I \times S \rightarrow O$. The three sets I, S and O are finite sets and made of inputs x, states s and outputs o, respectively. In the machine as illustrated in Figure 1, the transition function δ uses the set I of inputs as a first input and the set S of states as a second input to provide an output to the set S of states. The output faction λ uses the set I of inputs as a first input and the set S of states as a second input to provide a output to the set O of outputs. The transition faction δ, the output function λ and the set of states operate as a processing circuit PC. In operation, the machine FSM is initially in a predetermined state and input sequences from the set I of inputs are successively applied to the two factions. In response to each input sequence, the machine computes a state s from the input sequence, the current state and the transition faction δ while computing an output o from the input sequence, the current state ad the output faction λ and switching into the state s.

Also, it can be said that the machine FSM produces a sequence of n outputs in response to a sequence of n inputs. In the following example:

$$\begin{array}{lll} \{s_1, s_2\} \subseteq S & \text{and } \delta : (s_1, s_2) \rightarrow s_2 & \text{and } \lambda : (s_1, x_1) \rightarrow o_2 \\ \{x_1, x_2\} \subseteq I & \delta : (s_2, x_1) \rightarrow s_1 & \lambda : (s_2, x_1) \rightarrow o_2 \\ \{o_1, o_2\} \subseteq O & & \lambda : (s_1, x_2) \rightarrow o_1 \end{array}$$

then the machine FSM produces the sequence "o_2 o_2 o_1" from the sequence "x_1 x_1 x_2" when the state is s_1.

The invention relates to the problem of computation of the observable equivalence relation of a machine FSM. It will be assumed that the machine has the state s and produces a sequence of m outputs $O = (o_1, \ldots, o_m)$ in response to a sequence of m inputs $X = (x_1, \ldots, x_m)$. The observable equivalence is the equivalence for the states with regard to the produced outputs. Two states s and s' are said to be equivalent if, from states s and s', the machine always produces the same output sequence in response to the same input sequence of any length. Two states s and s' are said to be k-equivalent (where k is an integer ≥ 1) if they are equivalent for any sequence of a length $\leq k$. The set of equivalent (respectively k-equivalent) state pairs is denoted E (respectively E_k). We assume that for any $j \leq i$ the set E_i is included in E_j. In other words, if $i<j$: $E \subseteq E_j \subseteq E_i \subseteq S \times S$.

Furthermore, the invention is concerned with a data structure based on boolean factions. A boolean faction is of the type $f(x_1, \ldots, x_n) : \{0, 1\}^n \rightarrow \{0, 1\}$. An identity is known between the boolean faction f and the set of variables $X \subseteq \{0, 1\}^n : X = \{(x_1, \ldots, x_n) \mid f(x_1, \ldots, x_n) = 1\}$, so that every set can be represented by its characteristic function and both are denoted by the same symbol. Let ϕ be a boolean function and $v_1, v_2, \ldots, v_n$ its variable support. Also, let $\phi[v_i \leftarrow 0]$ be the formula in which the variable v_i is replaced by the

constant 0 and O an order on $v_1, v_2, ..., v_n$ here considered as $v_1 < v_2 < ... < v_n$. Then a graph currently named Shannon's tree of ϕ in the ordering O and denoted $A(\phi)$ can be built in compliance with the two following rules. First, the tree of constants are the leaves "true" and "false" and, second, the tree has a single root labelled by the variable v_1. From the root grow two branches, i. e. a branch aiming in the right direction on the tree of ϕ $[v_i \leftarrow 0]$ and a branch aiming in the left direction on the tree of $\phi[v_i \leftarrow 1]$. Two equivalent boolean factions have the same Shannon's tree modulo an ordering on the set of boolean variables and the choice of a variable support. From Shannon's tree $A(\phi)$ a boolean faction is commonly represented by Bryant's binary decision diagram currently named BDD of ϕ for the ordering O and described in publication (2) IEEE Transactions on Computers, C-35(8), August 1986, pages 677-691, R. E. Bryant: "Graph-Based Algorithms for Boolean Function Manipulation". In brief, as described in this publication, a BDD of ϕ for the ordering O is obtained from Shannon's tree by application until exhaustion of two reduction rules : (1) redundant node elimination and (2) equivalent subgraph share. The BDD is obtained regardless of the order of application of the two rules. As a result of redundant node elimination, the BDD has a stronger canonicity property than that of Shannon's trees since it does not refer to a variable support. Two equivalent factions for a predetermined ordering on the set of boolean variables have the same BDD. For example, $\neg v_1 \vee v_1$ and $\neg v_2 \vee v_2$ have the same BDD.

The transition faction is a vector $\delta : \{0, 1\}^n \rightarrow \{0, 1\}^k$. It is a vector of k boolean functions $\delta = (\delta_1, ..., \delta_k)$, with $\delta_i : \{0, 1\}^n \rightarrow \{0, 1\}$. In other words, $\delta(x_1, ..., x_n) = [\delta_1(x_1, ..., x_n), ..., \delta_k(x_1, ..., x_n)]$.

More generally, a completely specified finite state machine is a 5-tuple $M = (I, O, S, \delta, \lambda)$. The set $I = \{0, 1\}^k$ is the input space, the set $O = \{0, 1\}^\ell$ is the output space and the set $S = \{0, 1\}^n$ is the state space. Each state variable s_j is associated with a boolean faction δj from S x I to $\{0, 1\}$ and each output variable o_j is associated with a boolean function λ_j from S x I to $\{0, 1\}$. The vector $\delta = (\delta_j)$ is the transition faction from S x I to S and the vector $\lambda = (\lambda_j)$ is the output function from S x I to O. The symbols $s_1, ..., s_n$ are used to denote the boolean state variables and $x_1, ..., x_k$ to denote the boolean input variables. The vector $[s_1, ..., s_n]$ is written s as well. In other words,

$I = \{0, 1\}^k$ encoded on variables $x_1, ..., x_k$
$O = \{0, 1\}^\ell$ encoded on variables $o_1, ..., o_\ell$
$S = \{0, 1\}^n$ encoded on variables $s_1, ..., s_n$
$\delta(s_1, ..., s_n, x_1, ..., x_k) : \{0, 1\}^{n+k} \rightarrow \{0, 1\}^n = (\delta_1, ..., \delta_n)$
with $\delta_i(s_1, ..., s_n, x_1, ..., x_k) : \{0, 1\}^{n+k} \rightarrow \{0, 1\}$
$\lambda(s_1, ..., s_n, x_1, ..., x_k) : \{0, 1\}^{n+k} \rightarrow \{0, 1\}^\ell = (\lambda_1, ...\lambda_\ell)$
with $\lambda_i(s_1, ..., s_n, x_1, ..., x_k) : \{0, 1\}^{n+k} \rightarrow \{0, 1\}$.

In Figure 2 is illustrated the structure of a product finite state machine referenced PFSM and here-under also named product machine. It is an intuitive and sequential object as that illustrated in Figure 1. The product machine is similar to that illustrated in Figure 1 in which processing circuit PC is duplicated to have a second processing circuit PC' connected parallel to first processing circuit PC between set I of inputs and set O of outputs. More specifically, the first processing circuit PC comprises a first transition function δ, a first output faction λ and a first set S of states s while the second processing circuit PC' comprises a second transition function δ', a second output function λ' and a second set S' of states s'. The first and second transition factions and the first and second output factions are connected to set I of inputs and respective sets S, S' of states while the outputs of first and second output factions are connected to set O of outputs.

Let s and s' be a pair of states, they are equivalent if the processing circuits having the respective states s and s' produce same outputs in response to same inputs. The product machine is used to determine whether a pair of states s and s' is made of equivalent states. The determination is made by concatenating the machine of Figure 1. Let $S = \{0, 1\}^n$ be encoded by the variables $s_1, ..., s_n$. Let $s'_1, ..., s'_n$ be n variables not used in the original machine. We denote the subset of pairs by BDD's on $s_1, ..., s_n, s'_1, ..., s'_n$. For each boolean function δ_i, the function δ'_i is obtained by substituting the variables $s'_1, ..., s'_n$ for the variables $s_1, ..., s_n$, respectively. Similarly, for each boolean function λ_i, the function λ'_i is obtained by substituting the variables $s'_1, ..., s'_n$ for the variables $s_1, ..., s_n$, respectively. Then,

$\delta'_i(s'_1, ..., s'_n, x_1, ..., x_k) = \delta_i([s_1 \leftarrow s'_1], ..., [s_n \leftarrow s'_n])$
$\lambda'_i(s'_1, ..., s'_n, x_1, ..., x_k) = \lambda_i([s_1 \leftarrow s'_1], ..., [s_n \leftarrow s'_n])$

The transition function of the product machine is denoted $\Delta = (\delta_1, ..., \delta_n, \delta'_1, ..., \delta'_n)$ and $\Delta(s_1, ..., s_n, s'_1, ..., s'_n, x_1, ..., x_k) : \{0, 1\}^{n+n+k} \rightarrow \{0, 1\}^{n+n}$

The output function of the product machine is denoted $\Lambda = (\lambda_1, ..., \lambda_\ell, \lambda'_1, ..., \lambda'_\ell)$ and $\Lambda(s_1, ..., s_n, s'_1, ..., s'_n, x_1, ..., x_k) : \{0, 1\}^{n+n+k} \rightarrow \{0, 1\}^{\ell+\ell}$.

Thus, the product machine is a 5-tuple $M' = (I, [O \times O], [S \times S], \Delta, \Lambda)$.

As other well known definitions, let δ be a boolean faction and x a variable, the following notations are used:

$\forall x\delta = \delta(x \leftarrow 0) \wedge \delta(x \leftarrow 1)$
$\exists x\delta = \delta(x \leftarrow 0) \vee \delta(x \leftarrow 1)$

in which x can be a vector of boolean variables.

Also, let δ be a boolean function $\delta : \{0, 1\}^n \rightarrow \{0, 1\}^k$ and $P \subseteq \{0, 1\}^k$ then the reverse image is $\delta^{-1}(P) \subseteq \{0, 1\}^n = \{(x_1, ..., x_n) \mid \delta(x_1, ..., x_n) \in P\}$

Once the product machine is constructed, each state in the product machine actually represents a pair of states s, s' for the machine. The goal is to compute the BDD on the variables $s_1, ..., s_n, s'_1, ..., s'_n$, of the equivalent state pairs in which the first component is encoded on the variables $s_1, ..., s_n$ and the second component is encoded on the variables $s'_1, ..., s'_n$. In the prior art, an algorithm is used to get the equivalent state pairs. This algorithm consists in successively constructing the BDD's of the sets E_j until the fixpoint is reached. The fixpoint of a function f is the element $x : f(x) = x$. Here the fixpoint is reached when $E_n = E_{n+1} = E$ and is obtained by computing the E_j suite as follows:

$$E_1(s, s') = \forall x \bigwedge_{j \leq l} (\lambda_j(s, x) \Leftrightarrow \lambda'_j(s', x))$$

$E_n = E_1 \wedge (\forall x(\Delta^{-1}(E_{n-1}))$

This computation is based on a good variable ordering for the BDD's of the sets E_j, which must be compatible with the relation corroborated by the experimentation :

$\{s_1, s'_1\} < \{s_2, s'_2\} < ... < \{s_n, s'_n\}$

Intuitively, such an ordering makes the most of the two following facts : the sets E_j are graphs of equivalence relations and the original ordering of the sate space is $s_1 < s_2 < ... < s_n$.

The computation of the set E_1 requires only boolean operations on BDD's, that are quadratic operations, and an elimination of variables. Experimental tests stipulate that the construction of set E_1 is generally performed at a low cost when the above variable ordering is chosen. To compute the set E_n from E_{n-1}, we have to calculate the reverse image of E_{n-1} by vector Δ and to perform a universal elimination of the input variables. The reverse image computation can be performed using several methods. One involves the construction of the graph of the transition faction, but this graph cannot be built for very large machines FSM. Another method (called substitution method) consists in replacing each of variables $s_1, s'_1, ..., s_j, s'_j, ...$ by the corresponding $\delta_1, \delta'_1, ..., \delta_j, \delta'_j, ...$ There exist some implementations that perform simultaneously this substitution ad elimination of the input variables. An example of this prior method is described in the above-cited publication (1).

The computation of reverse image by the substitution method is made by an algorithm having a exponential complexity. Moreover, the vector Δ comprises 2n functions (δ, δ') and the reverse image is computed from the set E_{n-1} having 2n boolean variables. Thus the computation of $\Delta^{-1}(E_n)$ is a long time consuming step and requires a large memory space.

The present invention relates to a new method of computing $\Delta^{-1}(En)$, which corresponds to a new computation of the fixpoint reached when $E_n = E_{n+1} = E$. It overcomes the drawbacks of the prior art in allowing computation in a short time using a substantially less large memory space.

More generally, the invention provides a method of verification of a finite state sequential machine, comprising computing by means of a computer a set Y defined from two finite sets B and S encoded on boolean variables, a function $\delta : B \rightarrow S$ expressed by a vector of boolean functions, and a equivalence relation A on S, the set B being encoded on variables $s = s_1, ..., s_n$ and $x = x_1, ..., x_k$, in which x can be void, and Qx_i designating either $\exists x_i$ or $\forall x_i$, so that:

$Y = \{(s, s') \in (\exists x_1, ..., \exists x_k B)^2 \mid Qx_1, ..., Qx_k [(\delta(s, x), \delta(s', x)) \in A]\}$,

characterized in that the computation of set Y comprises the steps of (a) constructing in a canonical way, from the BDD of the graph of the equivalence relation A, the BDD of the graph of a total faction from S x S, named cross-section and denoted C(A), (b) constructing from the cross-section and the vector δ a new vector denoted $\delta^* = C(A) \circ \delta$, and (c) computing the pairs (s, s') such that $Qx1, \ldots Qxk .(\delta^*(s, x) = \delta(s', x))$.

More particularly, the method is used to compute a equivalence relation E of a finite state machine (FSM), the equivalence relation E being defined as the fixpoint of a monotonic suite $E_1, \ldots, E_n = E$, the computation of the equivalence relation E being made by successively construct the set E_n by use of the reverse image of the set E_{n-1} defined in terms of said set Y.

One result is an information support incorporating a computer program carrying out the method of the invention.

A second result is a tool of verification of a finite state sequential machine carrying out the method of the invention.

The objects and advantages of the invention will become clearly apparent from the following description of a preferred embodiment of the invention referring to the appended drawings. In the drawings :

Figure 1 is a block diagram of a finite state machine, and

Figure 2 is a block diagram of a product finite state machine.

The underlying idea of the present invention is based on the fact that the set $\Delta^{-1}(E_n)$ is the graph of an equivalence relation on S x S. To obtain set E_1, all the pairs of states having the same image by δ are to be found. To find them, the invention is directed to find from the set E_{n-1} a vector θ having the following property:

$$[\theta(s) = \theta(s')] \Leftrightarrow [(s, s') \in \Delta^{-1}(E_{n-1})]$$

With the vector θ can be constructed $\Delta^{-1}(E_n)$ in the same manner as set E_1 is built from output function λ.

The present invention relates to a method of computing the reverse image of the transition function of a product finite state machine from the set of n-1 equivalent states, that is a method of computing $\Delta^{-1}(E_{n-1})$, comprising the steps of:

(a) constructing in a canonical way, from the BDD of the graph of the relation E_{n-1}, the BDD of the graph of a total function from S x S, named cross-section and denoted $C(E_{n-1})$, and

(b) constructing from the cross-section and the vector δ a new vector denoted $\delta^{n-1} = C(E_{n-1}) \circ \delta$. It will be shown that the new vector has the desired properties of the above vector θ. Thus this vector is obtained by a modification of the transition function δ with the help of the set $C(E_{n-1})$. Although the reverse image computation algorithm is used, the sets whose reverse image is now computed are subsets of S (encoded by the variables $s_1, s_2, \ldots, s_n$) instead of the set E_{n-1} of state pairs (encoded by the variables $s_1, s_2, \ldots, s_n, s'_1, s'_2, \ldots, s'_n$) as in the prior method.

Then the method of the invention can comprise a third step (c) of computing the equivalent pairs of states with respect to the vector δ^{n-1}. These pairs are exactly the elements of $(\forall x(\Delta^{-1}(E_{n-1}))$. However, it is clear that the third step can be used to compute any existential or universal elimination of the variables in computing the reverse image by use of the method of the invention. In other words, instead of the step (c) used to compute the elements of $(\forall x(\Delta^{-1}(E_{n-1}))$ an alternative third step (c') can be used to compute the elements of $(\exists x(\Delta^{-1}(E_{n-1}))$.

The canonical cross-section $C(E_{n-1})$ defined in step (a) can be computed by using various methods. Here is presented a canonical way to obtain a graph of a function C(R) from a graph of a relation R. The function C(R) is called the cross-section of the relation R. The definition that will be given is a particular case of a method introduced in publication (3) Research Report, University of California, Berkeley, 1992, B. Lin and A. Richard Newton : "Implicit Manipulation of Equivalence Classes Using Binary Decision Diagrams", known under the name of compatible projection.

It will be assumed that all the elements of a set $\{0, 1\}^n$ are ordered by the increasing order of integers. For instance, the element (0, 0, ..., 1) represented by $(\neg s_1 \wedge \neg s_2 \wedge \ldots \wedge \neg s_{n-1} \wedge s_n)$ is the predecessor of (0, 0 ..., 1, 0) represented by $(\neg s_1 \wedge \neg s_2 \wedge \ldots \wedge s_{n-1} \wedge \neg s_n)$.

Definition 1 : Let B and C be two sets and ϕ a relation from B into C and let $\leq$ be a total order on C, then the cross-section of ϕ is the partial function $C(\phi)$ defined as :

$$C(\varphi) : \left| \begin{array}{l} B \rightarrow C \\ x \mapsto \min(\{y | (x, y) \in \varphi\}) \end{array} \right.$$

To compute the cross-section of relation E_{n-1}, the variables of the codomain are $s_1, ..., s_n$ and the variables for the domain are $s'_1, ..., s'_n$. From the BDD of E_{n-1} can be recursively constructed the BDD of $C(E_{n-1})$ applying the following transformation that remove from the BDD of E_{n-1} the pairs that are not in $C(E_{n-1})$, which is the following algorithm 1.

```
CrossSection(bdd) {
    if((bdd == true) || (bdd == false))
      return(bdd);

    if(bdd == [s_i, left, right] )
      return( [s_i, C(left), C(right)] );

    if(bdd == [s'_i, left, right] ) {
      aux = ¬∃s'_1 , ..., s'_n(left);
      return( [s'_i, C(left), C(right) ∧ aux] );

    }
}
```

This construction can be performed by a unique traversal of the graph of E_{n-1}, provided intermediate results are stored in the graph nodes.

Then, according to the second step (b) of the method of the invention, the composition $C(E_{n-1}) \circ \delta$ denoted δ^{n-1} is to be constructed. The construction could be made by using different methods. The following example is a method proceeding by separately modifying each of the original function δ_j of δ using the BDD of $C(E_{n-1})$.

The function δ is given as a vector of boolean functions $(\delta_1, ..., \delta_n)$ and the δ_j are on the state variables s_i and the input variables x_i. The vector $\delta^{n-1} = \delta_1{}^{n-1}, ..., \delta_n{}^{n-1})$ is built on the same variables, and we must have $\delta^{n-1}(s, x) = C(E_{n-1})\,\delta\,(s, x)$. Since E_{n-1} is an equivalence relation, it is trivial to verify that $C(E_{n-1})$ is a function defined everywhere. Consequently, the composition δ^{n-1} is defined everywhere, as δ is.

Let s be in S and K_i (s) denote the value of the component s_i of s. For example, $K_3([0, 1, 1, ..., 0])$ is 1. The function $\delta_j{}^{n-1}$ can be defined by the following equation, for j = 1 to n:

$$\delta_1{}^{n-1}(s, x) = K_j\,[C(E_{n-1})(\delta(s, x))]$$

Let us consider the BDD G of the graph of cross-section $C(E_{n-1})$. For j from 1 to n we construct the set

$$D_j = (\exists s'(\neg(s_j \Leftrightarrow s'_j) \wedge G)$$

$Dj \subseteq S$ includes all the states s such that $K_j(s) \neq K_j(G)$. The $\delta_j{}^{n-1}$ are obtained using the following theorem.

<u>Theorem 1</u> : Let F be the vector $C(E_{n-1})$, then $F \circ \delta = \delta^{n-1} = (\delta_1{}^{n-1}, ..., \delta_n{}^{n-1})$ can be defined as

$$\delta_j{}^{n-1}. = \neg(\delta_j \Leftrightarrow \delta^{-1}(D_j))$$

As a proof, it is checked that $\delta_j{}^{n-1}(s, x) = K_j\,[F(\delta(s, x))]$ by cases, depending on the values of $K_j(\delta(s, x))$ and $K_j\,[F(\delta(s, x))]$:

$K_j(\delta(s, x)) = 0$ and $K_j\,[F(\delta(s, x))] = 0$. Since $\delta(s, x)$ is not in D_j and (s, x) is not in $\delta^{-1}(D_j)$ and since (s, x) is not in δ, $\delta_j{}^{n-1}(s, x) = 0$.

$K_j(\delta(s, x)) = 0$ and $K_j\,[F(\delta(s, x))] = 1$. Since $\delta(s, x)$ is in D_j and (s, x) is not in $\delta^{-1}(D_j)$ and since $\delta(s, x) = 0$

we have $\delta_j^{n-1}(s, x) = 1$.
$K_j(\delta(s, x)) = 1$ and $K_j[F(\delta(s, x))] = 0$. Since $\delta(s, x)$ is in D_j and (s, x) is in $\delta^{-1}(D_j)$ and since (s, x) is in $\delta(s, x) = 0$, $\delta_j^{n-1}(s, x) = 0$.
$K_j(\delta(s, x)) = 1$ and $K_j[F(\delta(s, x))] = 1$. Since $\delta(s, x)$ is not in D_j and (s, x) is not in $\delta^{-1}(D_j)$ and since (s, x) is in δ, $\delta_j^{n-1}(s, x) = 0$.

The correctness of the composition will now be shown. It must be checked that the following equation holds:

$$[(s, s') \in \forall x(\Delta^{-1}(E_{n-1}))] \Leftrightarrow \forall x(\delta^{n-1}(s) = \delta^{n-1}(s'))$$

Proposition 1 : If ϕ is a transitive and symmetrical relation from B into B defined anywhere, then

$$\forall x, y((C(\phi)(x) = C(\phi)(y)) \Leftrightarrow ((x, y) \in \phi))$$

Proof:
$\Rightarrow$ Let $t = C(\phi)(x)$. By definition, two pairs (x, t) and (y, t) are in ϕ. By symmetry, (t, y) is in ϕ. Thus, by transitivity, (x, y) is in ϕ.
$\Leftarrow$ Suppose that $C(\phi)(x) < C(\phi)(y)$ and that $(x, y) \in \phi$. By symmetry, (x, y) is in ϕ. By transitivity, $(yC(\phi)(x))$ is in ϕ. Now, $(yC(\phi)(x)) \in \phi$ is inconsistent with the assumption $C(\phi)(y) < C(\phi)(x)$.

Theorem 2 : The composition $C(E_{n-1}) \circ \delta$ verifies the property of the equation to be shown,.
The statement results from Proposition 1.

$\forall x \in I\, [F(\delta(s, x)) = F(\delta(s', x))]$
$\Leftrightarrow$ $\forall x \in I\, [(\delta(s, x), \delta(s', x)) \in E_{n-1}]$ from Proposition 1
$\Leftrightarrow$ $\forall x \in I\, [(\delta(s, s', x)) \in E_{n-1}]$ from definition of δ
$\Leftrightarrow$ $(s, s') \in (\forall x(\Delta^{-1}(E_{n-1}))$ from a $\forall$-elimination of input variables

Thus, a new algorithm allows to compute the set E_n from E_{n-1}. The BDD is the graph of an equivalence relation on S x S. The set E_{n-1} is built on boolean variables $s_1, s_2, ..., s_n, s'_1, s'_2, ..., s'_n$ where s_i encodes the first component of pairs of S x S, the s'_i encoding the second component of pairs of S x S. First, the BDD of the graph of the canonical application of tins equivalence relation is built and named $C(E_{n-1})$. $C(E_{n-1})$ is built on the boolean variables $s_1, s_2, ..., s_n, s'_1, s'_2, ..., s'_n$, in which the s_i encode the first component of pairs of S x S and the s'_i encode the second component.

Let ϕ be a function of the type S x S having a graph $C(E_{n-1})$. From the BDD of $C(E_{n-1})$ and transition function δ is computed vector δ^{n-1} in accordance with the method of the invention. The vector δ^{n-1} is built by renaming the variables s into s'. Then is computed the set $\forall x(\delta^{n-1} \Leftrightarrow \delta'^{n-1})$. The set E_n is the intersection of this latter set built on the variables $s_1, s_2, ..., s_n, s'_1, s'_2, ..., s'_n$ and set E_1. Thus, the set $\forall x(\Delta^{-1}(E_{n-1})$ is built from δ^{n-1} in the same manner as E_1 is constructed from Λ. Finally, the set E_n is equal to $(\forall x(\Delta^{-1}(E_{n-1}) \wedge E_1)$.

The following second algorithm is given for the iterative step. This algorithm takes as input the BDD of E_{n-1} and the list of variables $(s'_1, s'_2, ..., s'_n)$ used for a copy of state variables and for the range of cross-section $C(E_{n-1})$. The returned result is the BDD of (E_{n-1}). Both E_n and E_{n-1} have variable support $s_1, s_2, ..., s_n, s'_1, s'_2, ..., s'_n$. The algorithm 2 is :

```
IterativeStep(E, s') {
        proj = CrossSection(E, s');
        for (j≤n) {
          dif   = ∃s'((¬s_j ⇔ s'_j) ∧ proj);
          ant   = δ^-1.(dif);
          case (j) {
            (1) {
               new_1  = δ_1 ⇔ ¬ant;
               new'_1 = new_1 [s ← s'];
               break;
            }

            [...]

            (n) {
               new_n  = δ_n ⇔ ¬ant;
               new'_n = new_n [s ← s'];
               break;
            }
          }
        }
        pred = ∀x ⋀_{j≤l} (new_j ⇔ new'_j);
        E_new = E_l ∧ pred;
        return(E_new);
}
```

Comparison have been made between the algorithm of the present invention and the implementation of the prior art algorithm. The method of the invention enables less space complexity due the simplification of reverse image computation and generally allows better results in time. The following table shows the features of the testing models : circuit name of the ISCAS89 - benchmarks -, number of variables (successively: inputs; outputs; states), number of vertices of all BDD's, number of equivalence classes and number of steps required for computation.

circuit name	variables	vertices	equivalence classes	steps
s344	9 ; 11 ; 15	164	18608	5
s1488	8 ; 19 ; 6	476	49	2
s298	3 ; 6 ; 14	117	8061	16
s382	3 ; 6 ; 21	174	608448	93

The following table show the results from the method of the invention compared with the prior method. CPU times are reported in seconds on an IBM RS-6000 Workstation.

circuit name	CPU time		memory	
	invention	prior art	invention	prior art
s344	1.9	21.0	0.9	2.2
s1488	1.0	12.7	0.6	1.7
s298	8.5	286.4	1.6	7.5
s382	480.6	2608.4	3.7	10.6

The vertices created for computation appear from the following table.

circuit name	vertices	
	invention	prior art
s344	27127	110314
s510	16358	64387
s1488	15935	97622
s298	78210	1372970
s382	2600773	>3500000

The method of the invention can be modified by a man skilled in the art to have a plurality of various embodiments. For example, a BDD can have several embodiments, in particular the so-called TDG (Typed Decision Graph) described for example in the book The Fusion between Hardware Design and Verification, G. Milne as publisher, North-Holland, part entitled "Original Concepts of PRIAM, An Industrial Tool for Efficient Formal Verification of Combinational Circuits". Any form of BDD can be used by the present invention. Furthermore, although the ordering of all the BDD's is assumed to be : $s_1 < s'_1 < s_2 < ... < s_n < s'_n$, another ordering can be used and the method can be modified accordingly. Also, although the description of the above embodiment of the invention has been directed to compute the set $\forall x(\Delta^{-1}(E_{n-1})$, it has been shown that the invention can be extended to any quantification of each variable of x.

Also, although the method of the invention is a solution to a problem raised by a finite state machine applied to hardware design and verification, it can be extended to be a method of verification of a sequential machine having finite states, since a finite state machine is software sequential machine. For example, the method can be used for protocol verification, control interface verification and more generally for the verification of software program portions having finite states. Also, it is clear that the invention can be applied for verification of a hardware sequential machine such as a sequencer and an automated machine having finite states. The meaning of word "verification" includes that of test which is to verify whether required conditions are satisfied.

Furthermore, although the above preferred embodiment shows that the method of the present invention uses an iterative computation from E_{n-1}, it is obvious that the present invention can be used without iterative computation and for any quantification of each variable.

As a result, the method of the invention can be defined as a method of verification of a finite state sequential machine, comprising computing by means of a data processing machine a set Y defined from two finite sets B and S encoded on boolean variables, a function $\delta : B \rightarrow S$ expressed by a vector of boolean functions and an equivalence relation A on S, the set B being encoded on variables $s = s_1, ..., s_n$ and $x = x_1, ..., x_k$, in which x can be void, and Qx_i designating either $\exists x_i$ or $\forall x_i$,so that:

$$Y = \{(s, s') \in (\exists x_1, ..., \exists x_k B)^2 \mid Qx_1, ..., Qx_k [(\delta(s, x), \delta(s', x)) \in A]\},$$

characterized in that the computation of set Y comprises the steps of (a) constructing in a canonical way, from the BDD of the graph of the equivalence relation A, the BDD of the graph of a total function from S x S, named cross-section ad denoted C(A), (b) constructing from the cross-section and the vector δ a new vector denoted $\delta^* = C(A) \circ \delta$, and (c) computing the pairs (s, s') such that Qx1, ...Qxk $.(\delta^*(s, x) = \delta(s', x))$. In this definition, the sets B and S are any sets which can be different from the above sets B and S, the variables s and x are any variables which can be different form the above variables s and x, the equivalence

relation A is any equivalence relation and the function δ can be another function than the transition function of a FSM. Since no iterative computation is made, the vector δ^* is used in lieu of δ^{n-1} in the iterative computation of the above embodiment. The equation for computation of Y is the algorithm of computation of a reverse image. Accordingly, the connection of this general definition with the method as described in the above embodiment can be made in defining the latter as a method used to compute an equivalence relation (E) of a finite state machine FSM, the equivalence relation (E) being defined as the fixpoint of a monotonic suite $E_1, ..., E_n = E$, the computation of the equivalence relation (E) being made by successively constructing the set E_n by use of the reverse image of E_{n-1} defined in terms of said set Y.

In the above description of the preferred embodiment has been shown that the canonical cross-section used in the above step (a) can be computed from the compatible projection using the following definition : let B and C be two sets and ϕ a relation from B into C ad let $\leq$ be a total order on C, then the cross-section of ϕ is the partial function $C(\phi)$ defined as :

$$C(\varphi) : \left| \begin{array}{l} B \rightarrow C \\ x \mapsto \min(\{y | (x, y) \in \varphi\}). \end{array} \right.$$

In this definition is assumed that the increasing ordering of the variables. Another ordering and definition could be used. For example, another ordering can be used in replacing in the above definition "let $\leq$ be a total order on C" by "let < be a total strict order on C. Also, a computation from another method than the compatible projection can be used.

From theorem 1 in the preferred embodiment can also be shown that in step (b) in the general definition of the present invention the vector δ^* is built from vector δ ad graph C(A), this graph being encoded on said variable $s = s_1, ..., s_n$ for the domain and on a variable $s' = s'_1, ..., s'_n$ for the codomain, while δ^* is built by using the sets $D_j = \exists s'(\neg(s_j \Leftrightarrow s'_j) \wedge C(A))$ and performing the reverse image computation uniquely for the sets Dj. The word "uniquely" is a consequence of the present invention.

The method can be carried out by a computer program of an information support such as a magnetic tape or disk and by a verification tool.

Claims

1. Method of verification of a finite state sequential machine, comprising computing by means of a data processing machine a set Y defined from two finite sets B and S encoded on boolean variables, a function $\delta : B \rightarrow S$ expressed by a vector of boolean functions, and an equivalence relation A on S, the set B being encoded on variables $s = s_1, ..., s_n$ and $x = x_1, ..., x_k$, in which x can be void, and Qx_i designating either $\exists x_i$ or $\forall x_i$, so that :

   $Y = \{(s, s') \in (\exists x_1, ..., \exists x_k B)^2 \mid Qx_1, ..., Qx_k [(\delta(s, x), \delta(s', x)) \in A]\}$,

   characterized in that the computation of set Y comprises the steps of (a) constructing in a canonical way, from the BDD of the graph of the equivalence relation A, the BDD of the graph of a total function from S x S, named cross-section and denoted C(A), (b) constructing from the cross-section ad the vector δ a new vector denoted $\delta^* = C(A) \circ \delta$, and (c) computing the pairs (s, s') such that Qx1, ...Qxk .-($\delta^*(s, x) = \delta(s', x)$).

2. Method according to claim 1, characterized in that it is used to compute an equivalence relation E of a finite state machine (FSM), the equivalence relation E being defined as the fixpoint of a monotonic suite $E_1, ..., E_n = E$, the computation of the equivalence relation E being made by successively constructing the set E_n by use of the reverse image of the set E_{n-1} defined in terms of said set Y and corresponding to $\Delta^{-1}(E_{n-1})$.

3. Method according to claim 1 or 2, characterized in that said step (a) is computed from the compatible projection in which is used the following definition : let B and C be two sets and ϕ a relation from B into C and let < be a total strict order on C, then the cross-section of ϕ is the partial function $C(\phi)$ defined as :

$$C(\varphi) : \left| \begin{array}{l} B \rightarrow C \\ x \mapsto \min(\{y|(x, y) \in \varphi\}). \end{array} \right.$$

4. Method according to claim 1 or 2, characterized in that said step (a) is computed from the compatible projection in which is used the following definition: let B and C be two sets and ϕ a relation from B into C and let $\leq$ be a total order on C, then the cross-section of ϕ is the partial function $C(\phi)$ defined as:

$$C(\varphi) : \left| \begin{array}{l} B \rightarrow C \\ x \mapsto \min(\{y|(x, y) \in \varphi\}) \end{array} \right.$$

and the order is selected to be a successive increasing order.

5. Method according to anyone of claims 1 to 4, characterized in that in the step (b) the vector δ^* is built from the vector δ and the graph C(A), this graph being encoded on said variable $s = s_1, ..., s_n$ for the domain and on a variable $s' = s'_1, ..., s'_n$ for the codomain, and δ^* is also built by using the sets $D_j = \exists s' (\neg(s_j \Leftrightarrow s'_j) \wedge C(A))$ and performing the reverse image computation uniquely for the sets Dj.

6. Method according to anyone of claims 2 to 5, characterized in that said step (b) comprises constructing from said cross-section and said vector δ a new vector $\delta^{n-1} = C(E_{n-1}) \circ \delta$.

7. Method according to claim 6, characterized in that it further comprises a step (c) of computing the equivalent pairs of states with respect to the vector δ^{n-1} to have the pairs of $(\forall x \Delta^{-1}(E_{n-1}))$.

8. Information support incorporating a computer program, characterized in that the program carries out the method defined by anyone of claims 1 to 7.

9. Tool of verification of a finite state sequential machine, characterized in that it carries out the method defined in anyone of claims 1 to 7 or the information support of claim 8.

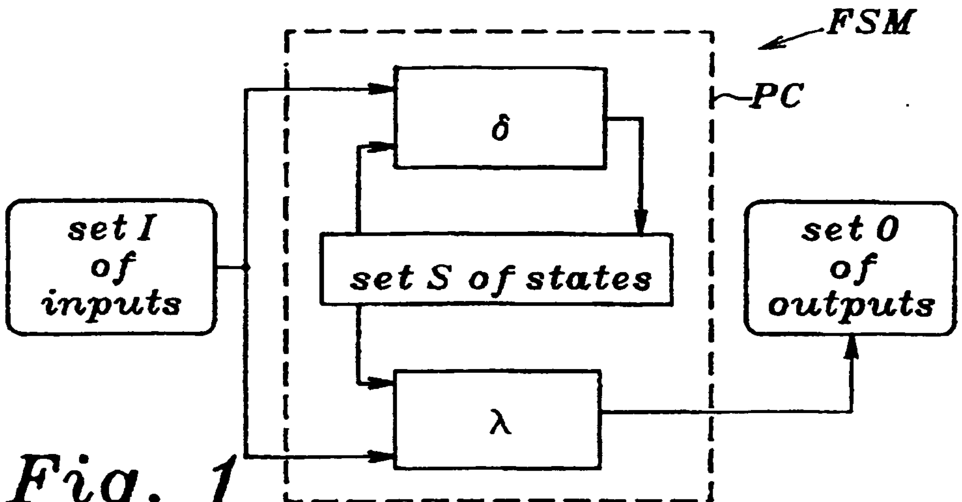
FSM
PC
δ
set I of inputs
set S of states
set O of outputs
λ

Fig. 1

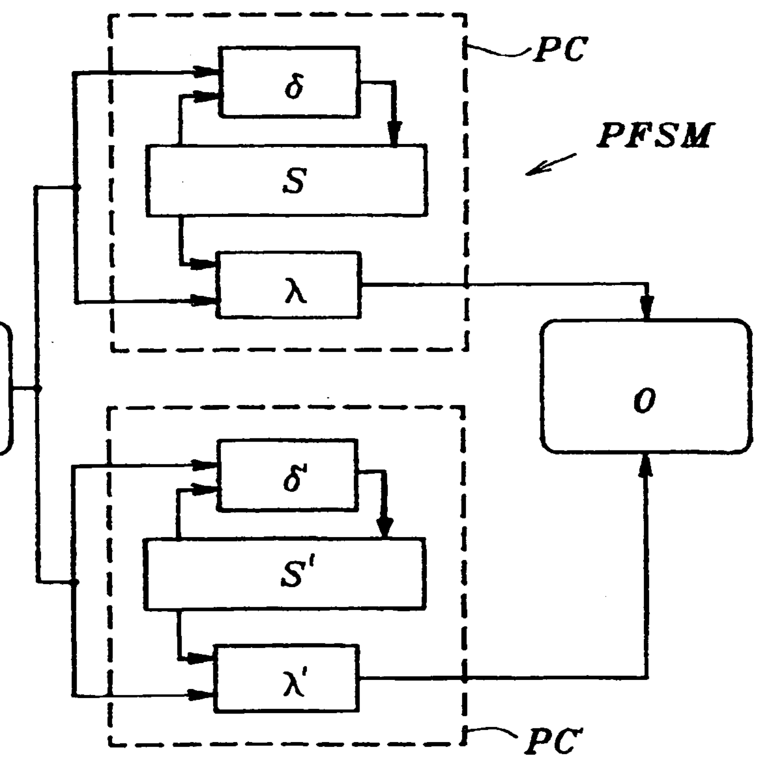
PC
δ
PFSM
S
λ
I
O
δ'
S'
λ'
PC

Fig. 2